CI1055: Algoritmos e Estruturas de Dados I

Profs. Drs. Marcos Castilho, Bruno Müller Jr, Carmem Hara

Departamento de Informática/UFPR

30 de julho de 2020

Resumo

Cálculo de séries

Objetivos da aula

- Mostrar como implementar séries, em particular as *Séries de Taylor*

Motivação

- Muitas funções importantes podem ser calculadas usando-se Séries de Taylor
- Exemplos: $e^x, sen(x), cos(x), \ldots$

Do Cálculo Diferencial e Integral

- Uma série de Taylor tem a forma:

$$f(x) = \sum_{n=0}^{\infty} a_n (x - a)^n,$$

onde

$$a_n = \frac{f^{(n)}(a)}{n!}$$

- O cálculo é feito em torno do ponto $x = a$
- $f^{(n)}$ é a n-ésima derivada de f.

Exemplos

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!}$$

$$ln(1+x) = \sum_{n=0}^{\infty} \frac{(-1)^n x^{n+1}}{n+1}, |x| < 1$$

$$sen(x) = \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n+1}}{(2n+1)!}$$

Exemplos

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \frac{x^5}{5!} + \dots$$

$$ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \frac{x^5}{5} - \dots$$

$$sen(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} - \dots$$

Motivação

- Computar estes cálculos não é tão simples quanto parece
- Os cálculos podem não convergir
- Vamos usar a técnica do acumulador, dentre outras
- Vamos iniciar com uma série bem simples
- Pretendemos mostrar como se calcula a função $sen(x)$ até o final desta aula

A série Harmônica

$$H_n = \sum_{n=0}^{\infty} \frac{1}{n} = 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \frac{1}{5} + \dots$$

- A série Harmônica diverge
- Não podemos computar até o infinito
- Vamos assumir que o cálculo será feito até o n-ésimo termo, sendo n pornecido pelo usuário

Cálculo computacional de séries

- Baseada na técnica do acumulador
- O segredo é perceber como um termo pode ser obtido a partir do anterior
- Ou de modo equivalente: como transformar um terno no próximo
- Se isto for bem resolvido então a implementação é simples, pois o uso da técnica do acumulador é bastante simples

Computação da série Harmônica

$$H_n = 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \frac{1}{5} + \ldots + \frac{1}{n}$$

- Observar os termos:
  - uma fração com um *numerador* e um *denominador*
- Observar a diferença entre os termos:
  - todos os numeradores são iguais a 1
  - o primeiro denominador é 1
  - um denominador é obtido do anterior somando-se 1

Computação da série Harmônica

$$H_n = 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \frac{1}{5} + \ldots + \frac{1}{n}$$

- Numerador: manter constante igual a 1
- Inicialização do denominador: `den:= 1`
- Padrão repetitivo do denominador: `den:= den + 1`
- Critério de parada: quando o contador atingir n

Implementação da série Harmônica

$$H_n = 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \frac{1}{5} + \ldots + \frac{1}{n}$$

```
program serie_harmonica;
var cont, num, den, n: longint;
    soma: real;
begin
    read (n);
    cont:= 1;
    num:= 1;
    den:= 1;
    soma:= 0;
    while cont <= n do
    begin
        soma:= soma + num/den;
        cont:= cont + 1;
        den:= cont;
    end;
    writeln (soma:0:5);
end.
```

Cálculo do seno

$$sen(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} - \dots$$

- Parece bem mais complicado, não?
- Numeradores:
  - são potências de x, sempre ímpares
- Denominadores:
  - são fatoriais de números ímpares
- O sinal dos termos se invertem a cada termo

Vamos fazer em etapas, tentando resolver um problema de cada vez

Comparando seno com harmônica

$$H_n = 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \frac{1}{5} + \ldots + \frac{1}{n}$$

$$sen(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} - \ldots$$

- Saber somar termos é simples
- Como dito, o segredo é transformar um termo no próximo
- Vamos implementar o seno progressivamente, partindo da série harmônica, até chegarmos ao seno

Uma série mais simples do que o seno

$$H_n = 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \frac{1}{5} + \ldots + \frac{1}{n}$$

$$F1_n = 1 + \frac{1}{2!} + \frac{1}{3!} + \frac{1}{4!} + \frac{1}{5!} + \ldots + \frac{1}{n!}$$

- A diferença entre elas é o denominador:
  - na primeira, são números inteiros sequenciais
  - na segunda, são fatoriais de inteiros sequenciais

Cálculo de $f1$

Harmônica

```
begin
    read (n);
    cont:= 1;
    num:= 1;
    den:= 1;
    soma:= 0;
    while cont <= n do
    begin
        soma:= soma + num/den;
        cont:= cont + 1;
        den:= cont;
    end;
    writeln (soma:0:5);
end.
```

$F1$

```
begin
    read (n);
    cont:= 1;
    num:= 1;
    den:= 1;
    soma:= 0;
    while cont <= n do
    begin
        soma:= soma + num/den;
        cont:= cont + 1;
        den:= den * cont;
    end;
    writeln (soma:0:5);
end.
```

Trocando o sinal

$$F1_n = 1 + \frac{1}{2!} + \frac{1}{3!} + \frac{1}{4!} + \frac{1}{5!} + \ldots + \frac{1}{n!}$$

$$F2_n = 1 - \frac{1}{2!} + \frac{1}{3!} - \frac{1}{4!} + \frac{1}{5!} - \ldots + (-1)^n \frac{1}{n!}$$

- A diferença entre elas é o sinal dos termos
  - na primeira, são todos positivos
  - na segunda, são invertidos a cada termo

Cálculo de $f2$

$F1$

```
begin
    read (n);
    cont:= 1;
    num:= 1;
    den:= 1;
    soma:= 0;

    while cont <= n do
    begin
        soma:= soma + num/den;

        cont:= cont + 1;
        den:= den * cont;
    end;
    writeln (soma:0:5);
end.
```

$F2$

```
begin
    read (n);
    cont:= 1;
    num:= 1;
    den:= 1;
    soma:= 0;
    sinal:= 1;
    while cont <= n do
    begin
        soma:= soma + sinal*num/den;
        sinal:= -sinal;
        cont:= cont + 1;
        den:= den * cont;
    end;
    writeln (soma:0:5);
end.
```

Cuidando do numerador

$$F2_n = 1 - \frac{1}{2!} + \frac{1}{3!} - \frac{1}{4!} + \frac{1}{5!} - \ldots + (-1)^n \frac{1}{n!}$$

$$F3_n = x - \frac{x^2}{2!} + \frac{x^3}{3!} - \frac{x^4}{4!} + \frac{x^5}{5!} - \ldots + (-1)^n \frac{x^n}{n!}$$

- A diferença entre elas é o numerador
  - na primeira, são todos iguais a 1
  - na segunda, são potencias progressivas de x

Cálculo de $f3$

$F2$

```
begin
    read (n);
    cont:= 1;
    num:= 1;
    den:= 1;
    soma:= 0;
    sinal:= 1;
    while cont <= n do
    begin
        soma:= soma + sinal*num/den;
        sinal:= -sinal;
        cont:= cont + 1;

        den:= den * cont;
    end;
    writeln (soma:0:5);
end.
```

$F3$

```
begin
    read (n, x);
    cont:= 1;
    num:= x;
    den:= 1;
    soma:= 0;
    sinal:= 1;
    while cont <= n do
    begin
        soma:= soma + sinal*num/den;
        sinal:= -sinal;
        cont:= cont + 1;
        num:= num * x;
        den:= den * cont;
    end;
    writeln (soma:0:5);
end.
```

Alterando o numerador

$$F3_n = x - \frac{x^2}{2!} + \frac{x^3}{3!} - \frac{x^4}{4!} + \frac{x^5}{5!} - \ldots + (-1)^n \frac{x^n}{n!}$$

$$F4_n = x - \frac{x^3}{2!} + \frac{x^5}{3!} - \frac{x^7}{4!} + \frac{x^9}{5!} - \ldots + (-1)^n \frac{x^{(2*n+1)}}{n!}$$

- A diferença entre elas é o numerador
  - na primeira, são potencias progressivas de x
  - na segunda, são potencias ímpares progressivas de x

Cálculo de $f4$

$F3$

```
begin
    read (n, x);
    cont:= 1;
    num:= x;
    den:= 1;
    soma:= 0;
    sinal:= 1;
    while cont <= n do
    begin
        soma:= soma + sinal*num/den;
        sinal:= -sinal;
        cont:= cont + 1;
        num:= num * x;
        den:= den * cont;
    end;
    writeln (soma:0:5);
end.
```

$F4$

```
begin
    read (n, x);
    cont:= 1;
    num:= x;
    den:= 1;
    soma:= 0;
    sinal:= 1;
    while cont <= n do
    begin
        soma:= soma + sinal*num/den;
        sinal:= -sinal;
        cont:= cont + 1;
        num:= num * x * x;
        den:= den * cont;
    end;
    writeln (soma:0:5);
end.
```

Finalmente, $sen(x)$

$$F4_n = x - \frac{x^3}{2!} + \frac{x^5}{3!} - \frac{x^7}{4!} + \frac{x^9}{5!} - \ldots + (-1)^n \frac{x^{(2*n+1)}}{n!}$$

$$sen(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} - \ldots + (-1)^n \frac{x^{(2*n+1)}}{(2*n+1)!}$$

- A diferença entre elas é o denominador:
  - na primeira, são fatoriais de n progressivas
  - na segunda, são fatoriais de n de ímpares progressivos

Cálculo de $sen(x)$

$F4$

```
begin
    read (n, x);
    cont:= 1;
    num:= x;
    den:= 1;
    soma:= 0;
    sinal:= 1;
    while cont <= n do
    begin
        soma:= soma + sinal*num/den;
        sinal:= -sinal;
        cont:= cont + 1;
        num:= num * x * x;
        den:= den * cont;
    end;
    writeln (soma:0:5);
end.
```

$sen(x)$

```
begin
    read (n, x);
    cont:= 1;
    num:= x;
    den:= 1;
    soma:= 0;
    sinal:= 1;
    while cont <= n do
    begin
        soma:= soma + sinal*num/den;
        sinal:= -sinal;
        den:= den * (2*cont)*(2*cont+1);
        num:= num * x * x;
        cont:= cont + 1;
    end;
    writeln (soma:0:5);
end.
```

Resumo

- Calcular séries não é complicado
- Conforme explicado, basta perceber como o próximo termo é relacionado com o atual
- Percebendo esta diferença, implementa-se com tranquilidade

Resumo

- Por exemplo:
  - numerador: multiplica-se por x^2 para obter o próximo
  - denominador: aprende-se a fazer o fatorial de ímpares sucessivos

Fim do tópico

- este material está no livro no capítulo 7, seção 7.8

Licença

- Slides feitos em LaTeX usando beamer
- Licença

  Creative Commons Atribuição-Uso Não-Comercial-Vedada a Criação de Obras Derivadas 2.5 Brasil License.http://creativecommons.org/licenses/by-nc-nd/2.5/br/

  Creative Commons Atribuição-Uso Não-Comercial-Vedada a Criação de Obras Derivadas 2.5 Brasil License.http://creativecommons.org/licenses/by-nc-nd/2.5/br/

CI1055: Algoritmos e Estruturas de Dados I

Profs. Drs. Marcos Castilho, Bruno Müller Jr, Carmem Hara

Departamento de Informática/UFPR

30 de julho de 2020

Resumo

Maior segmento crescente

Objetivos da aula

- Apresentar uma interessante combinação de várias técnicas para resolver um problema nada trivial.

Motivação

- Os problemas agora se tornam bem complexos
- Exigem combinações das técnicas elementares
- Exige bastante raciocínio
- Usar técnica de resolver subproblemas ajuda!

Maior segmento crescente

Problema: Dada uma sequência de n números naturais, imprimir o valor do comprimento do segmento crescente de tamanho máximo dentre os números lidos. A sequência é terminada em zero, que não deve ser processado.

Exemplo 1:

$\boxed{5,\ 10,\ 3,\ 2,\ 4,\ 7,\ 9,\ 8,\ 5,\ \ldots}$

Resposta: 4, para o segmento 2, 4, 7, 9.

Exemplo 2:

$\boxed{10,\ 8,\ 7,\ 5,\ 2,\ \ldots}$

Resposta: 1, todos os segmentos crescentes são unitários.

Quais são os subproblemas

Problema: Dada uma sequência de n números naturais, imprimir o valor do comprimento do segmento crescente de tamanho máximo dentre os números lidos. A sequência é terminada em zero, que não deve ser processado.

- ler os números da entrada;
- para cada número, descobrir se estamos em um segmento crescente ou não;
- memorizar qual é o tamanho do segmento crescente "da vez";
- saber qual é o tamano do segmento crescente de tamanho máximo.
- podem haver outros. . .

Lendo os números da entrada

A sequência é terminada em zero, que não deve ser processado.

```
begin
    read (n);
    while n <> 0 do
    begin
        (* faz algo *)
        read (n);
    end;
end.
```

Saber se uma subsequência é crescente

É preciso lembrar do número anterior.

*Rascunho*1

```
begin
    read (n);
    while n <> 0 do
    begin
        (* faz algo *)
        read (n);
    end;
end.
```

Rascunho 2

```
begin
    read (n);
    while n <> 0 do
    begin
        n_anterior:= n;
        read (n);
        if n > n_anterior then
            (* seq crescente *)
        else
            (* acabou sequencia *)
    end;
end.
```

Problemas

A sequência pode ter tamanho 1. O primeiro pode ser negativo.

*Rascunho*2

```
begin
    read (n);
    while n <> 0 do
    begin
        n_anterior:= n; (* <--- *)
        read (n);
        if n > n_anterior then
            (* seq crescente *)
        else
            (* acabou sequencia *)
    end;
end.
```

Rascunho 3

```
begin
    read (n);
    n_anterior:= n;
    if n <> 0 then
    begin
        read (n);
        while n <> 0 do
        begin
            if n > n_anterior then
                (* seq crescente *)
            else
                (* acabou sequencia *)
            read (n);
        end;
    end;
end.
```

Memorizar o tamanho do segmento crescente atual

- Tem que introduzir um contador

Memorizar o tamanho do segmento crescente atual

Introduzindo um contador

*Rascunho*3

```
begin
    read (n);
    n_anterior:= n;
    if n <> 0 then
    begin
        read (n);
        while n <> 0 do
        begin
            if n > n_anterior then
                (* seq crescente *)
            else
                (* acabou sequencia *)
            read (n);
        end;
    end;
end.
```

Rascunho 4

```
begin
    read (n);
    n_anterior:= n;
    tam:= 0;
    if n <> 0 then
    begin
        read (n);
        tam:= tam + 1;
        while n <> 0 do
        begin
            if n > n_anterior then
                tam:= tam + 1
            else
                tam:= 1;
            n_anterior:= n;
            read (n);
        end;
    end;
end.
```

Qual é o tamanho do maior segmento?

- Usar a técnica de definir a priri e depois corrigir!

Calculando a maior sequência

Definir a priori e depois corrigir

Rascunho4

```
begin
    read (n);
    n_anterior:= n;
    tam:= 0;
    if n <> 0 then
    begin
        read (n);
        tam:= tam + 1;
        while n <> 0 do
        begin
            if n > n_anterior then
                tam:= tam + 1
            else
                tam:= 1;
            n_anterior:= n;
            read (n);
        end;
    end;
end.
```

Rascunho 5

```
begin
    read (n);
    n_anterior:= n;
    tam:= 0;
    maior_tam:= tam;
    if n <> 0 then
    begin
        read (n);
        tam:= tam + 1;
        while n <> 0 do
        begin
            if n > n_anterior then
                tam:= tam + 1
            else
            begin
                if tam > maior_tam then
                    maior_tam:= tam;
                tam:= 1;
            end;
            n_anterior:= n;
            read (n);
        end;
        writeln (maior_tam);
    end;
end.
```

Problema

- Se o maior segmento crescente for o último
- Neste caso, o tamanho deste não foi comparado!

Versão final

```
begin
    read (n);
    n_anterior:= n;
    tam:= 0;
    maior_tam:= tam;
    if n <> 0 then
    begin
        read (n);
        tam:= tam + 1;
        while n <> 0 do
        begin
            if n > n_anterior then
                tam:= tam + 1
            else
            begin
                if tam > maior_tam then
                    maior_tam:= tam;
                tam:= 1;
            end;
            n_anterior:= n;
            read (n);
        end;
        if tam > maior_tam then
            maior_tam:= tam;
        writeln (maior_tam);
    end;
end.
```

Em resumo. . .

- Este não é um problema trivial!
- Pudemos resolvê-lo pela divisão em subproblemas
- Cada subproblema pode ser resolvido com mais foco e corretude

Fim do tópico

- este material está no livro no capítulo 7, seção 7.9

Licença

- Slides feitos em LaTeX usando beamer
- Licença

  Creative Commons Atribuição-Uso Não-Comercial-Vedada a Criação de Obras Derivadas 2.5 Brasil License.`http://creativecommons.org/licenses/by-nc-nd/2.5/br/`

  Creative Commons Atribuição-Uso Não-Comercial-Vedada a Criação de Obras Derivadas 2.5 Brasil License.`http://creativecommons.org/licenses/by-nc-nd/2.5/br/`

CI1055: Algoritmos e Estruturas de Dados I

Profs. Drs. Marcos Castilho, Bruno Müller Jr, Carmem Hara

Departamento de Informática/UFPR

30 de julho de 2020

Resumo

Primos entre sí

Objetivos da aula

- Mostrar como integrar subproblemas já resolvidos em problemas novos que evidentemente dependem do subproblema.
- Motivar os estudantes para o importante próximo capítulo.

Números primos entre sí

Problema: Imprimir todos os pares de números (a, b) que são primos entre si para todo $2 \leq a \leq 100$ e $a \leq b \leq 100$.

- Definição: dois números naturais A e B são ditos *primos entre sí* se $MDC(A, B) = 1$.

Pseudo-código que resolve o problema

```
begin
    read (a, b);
    if ''a e b sao primos entre si'' then
        writeln ('SIM')
    else
        writeln ('NAO');
end.
```

- Obviamente resta calcular se *a* e *b* são primos entre si
- Pela definição, basta calcular o MDC entre eles
- Isto pode ser feito pelo conhecido, famoso e eficiente algoritmo de Euclides!

Algoritmo de Euclides.

```
program mdcporeuclides;
var a, b, resto: integer;
(* funciona para entradas nao nulas *)
begin
    read (a,b);
    resto:= a mod b;
    while resto <> 0 do
    begin
        a:= b;
        b:= resto;
        resto:= a mod b;
    end;
    writeln ('mdc = ', b);
end.
```

Programa para primos entre sí

```
program primosentresi;
var a, b, resto: integer;

begin
    read (a,b);
    if (a <> 0) and (b <> 0) then
    begin
        resto:= a mod b;
        while resto <> 0 do
        begin
            a:= b;
            b:= resto;
            resto:= a mod b;
        end;
        if b = 1 then
            writeln ('SIM')
        else
            writeln ('NAO');
    end;
end.
```

Em resumo

- Pode-se usar um código pronto que resolve um subproblema como parte da solução de outro problema
- Em breve, poderemos fazer o programa principal assim:

```
begin
    read (a, b);
    if mdc(a, b) then
        writeln ('SIM')
    else
        writeln ('NAO');
end.
```

- De maneira similar ao uso de uma função já disponível no compilador,
- Como por exemplo, `sqrt`, `sin`, `cos`, ...

Fim do tópico

- este material está no livro no capítulo 7, seção 7.10

Licença

- Slides feitos em LaTeX usando beamer
- Licença

  Creative Commons Atribuição-Uso Não-Comercial-Vedada a Criação de Obras Derivadas 2.5 Brasil License.http://creativecommons.org/licenses/by-nc-nd/2.5/br/

  Creative Commons Atribuição-Uso Não-Comercial-Vedada a Criação de Obras Derivadas 2.5 Brasil License.http://creativecommons.org/licenses/by-nc-nd/2.5/br/

CI1055: Algoritmos e Estruturas de Dados I

Profs. Drs. Marcos Castilho, Bruno Müller Jr, Carmem Hara

Departamento de Informática/UFPR

30 de julho de 2020

Resumo

Números primos

Objetivos da aula

- Desenvolver um dos mais completos programas desta fase do curso
- Praticamente integrando diversas técnicas básicas
- Apresentar o programa com o maior número de linhas até aqui
- Discutir princípios de eficiência dos algorimos

Números primos

- Um número natural positivo A é dito primo se não possui divisores próprios, isto é, não existem divisores d no intervalo $1 \leq d \leq A - 1$.

Subproblemas

- Se soubermos quantos divisores um dado número *a* possui a solução ficaria trivial, não?

```
begin
    read (n);
    num_div_proprios:= ``calcula o numero de divisores proprios de
         n'';
    if num_div_proprios = 0 then
        writeln ('SIM')
    else
        writeln ('NAO');
end.
```

Rascunho 1

```
program primos_v0;
var n, i, cont: longint;
begin
    read (n);
    cont:= 0;
    i:= 2;
    while i <= n-1 do
    begin
        if n mod i = 0 then
            cont:= cont + 1;
        i:= i + 1;
    end;
    if cont = 0 then
        writeln ('SIM')
    else
        writeln ('NAO')
end.
```

Problema

$\boxed{\text{Ineficiente!}}$

- Executa $n - 2$ iterações sempre
- Poderia parar se achar algum divisor!!!

Rascunho 2: para no primeiro divisor encontrado

```
program primos_v1;
var n, i, cont: longint; eh_primo: boolean;
begin
    read (n);
    cont:= 0;
    eh_primo:= true;
    i:= 2;
    while eh_primo and (i <= n-1) do
    begin
        if n mod i = 0 then
            eh_primo:= false;
        i:= i + 1;
    end;
    if eh_primo then
        writeln ('SIM')
    else
        writeln ('NAO')
end.
```

Ainda pode ser melhor

$\boxed{\text{Ineficiente!}}$

- É um pouquinho melhor do que o primeiro
- Mas, no pior caso, ainda executa $n - 2$
- Isso ocorre quando n é primo

Como melhorar?

- Só existe um único número par que é primo, o 2
- Todos os outros primos são ímpares
- Então vamos testar se n é par. Se for, resta saber se é o 2 ou outro par qualquer
- Depois resta testar todos os ímpares
- Assim, no pior caso, temos um primo ímpar e o algoritmo *só* vai iterar na ordem de metade das vezes

Rascunho 3: separa pares de ímpares

```
program primos_v2;
var n, i, cont: longint; eh_primo: boolean;
begin
    read (n);
    cont:= 0;
    if n mod 2 = 0 then (* eh par *)
        if n = 2 then eh_primo:= true
        else eh_primo:= false
    else (* eh impar *)
    begin
        eh_primo:= true;
        i:= 3;
        while eh_primo and (i <= n-1) do
        begin
            if n mod i = 0 then
                eh_primo:= false;
            i:= i + 2;
        end;
    end;
    if eh_primo then writeln ('SIM')
    else writeln ('NAO')
end.
```

Pode ser *BEM* melhor

$\boxed{\text{Ineficiente!}}$

- É, de novo, um pouquinho melhor do que o anterior
- Mas, no pior caso, ainda executa $\frac{n-2}{2}$
- Matematicamente, dado um número n, o maior divisor que ele pode ter é $\sqrt{n}$

Versão final: vai até $\sqrt{n}$

```
program primos_final;
var n, i, cont: longint; eh_primo: boolean;
begin
    read (n);
    cont:= 0;
    if n mod 2 = 0 then (* eh par *)
        if n = 2 then eh_primo:= true
        else eh_primo:= false
    else (* eh impar *)
    begin
        eh_primo:= true;
        i:= 3;
        while eh_primo and (i <= sqrt(n)) do
        begin
            if n mod i = 0 then
                eh_primo:= false;
            i:= i + 2;
        end;
    end;
    if eh_primo then writeln ('SIM')
    else writeln ('NAO')
end.
```

Comparação das funções

n	$\frac{n}{2}$	$\sqrt{n}$
1000000	500000	1000
1000000000	500000000	31622
1000000000000	500000000000	1000000

- Para n da ordem de 10^{12}
  - a versão do rascunho 3 vai levar muito tempo
  - a versão final seis ordens de grandeza a menos

Conclusão

- O melhor teria sido analisar completamente o problema *antes* de fazer qualquer implementação
- Neste caso, felizmente, a adaptação foi simples
- Existem casos em que toda a implementação tem que ser jogada fora para implementar outra, totalmente diferente
- A busca pela eficiência é um diferencial em Ciência da Computação!

Conclusão

- Com isso terminamos o capítulo 7
- Passaremos às questões de modularidade
- Depois, finalmente, às estruturas de dados

Exercícios

- Fazer os exercícios 1 a 17 da seção 7.12 do livro [1]
- Os exercícios de 18 a 22 são desafios

[1] `http://www.inf.ufpr.br/cursos/ci055/livro_alg1.pdf`

Fim do tópico

- este material está no livro no capítulo 7, seção 7.11

Licença

- Slides feitos em LaTeX usando beamer
- Licença

  Creative Commons Atribuição-Uso Não-Comercial-Vedada a Criação de Obras Derivadas 2.5 Brasil License.`http://creativecommons.org/licenses/by-nc-nd/2.5/br/`

  Creative Commons Atribuição-Uso Não-Comercial-Vedada a Criação de Obras Derivadas 2.5 Brasil License.`http://creativecommons.org/licenses/by-nc-nd/2.5/br/`